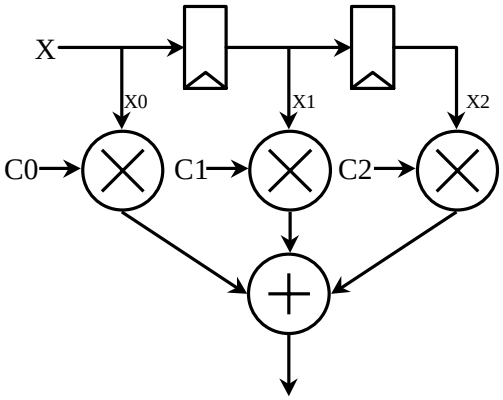
FIR Filter

- **3 TAP FIR filter**



1-1

•This architecture represents the direct-form implementation of a 3-tap Finite Impulse Response filter. The number of “taps” is equal to the number of multiplications.

•A history of the input sample, X, is captured and feeds a merged sum of products circuit where each data sample is multiplied by a corresponding coefficient, C0, C1, C2.

•This type of architecture is used for high sample rate filters. It provides high sample rate because it computes the sum of all products in parallel.

•For lower sample rate filters fewer multipliers can be used. A multiplier-accumulate type architecture can be used if the sample rate is low enough (one multiplication per clock cycle).

FIR Filter

- **FIR Filter**
  - Works With Any Width/Format Coeff and Data
  - Works With Any Number of Taps

```
function FIR (Z,X,repl(i,fnArgs()-2,“,”){C{i}});
    integer taps=fnArgs()-2;
    input X,repl(i,taps,“,”){C{i}};
    wire XX,repl(i,taps,“,”){X{i}};
    XX = sreg(X,taps-1,repl(i,taps,“,”){X{i}});
    output Z = repl(i,taps,”+”){X{i}*C{i}};
endfunction
```

```
X=FIR(D1,C0,256,C2);      // 3-tap, some constant coeffs
Y=FIR(A,C0,C1,C2,C3);     // 4-taps
Z=FIR(D1,-5,10,-5);       // 3-taps all constant coeffs
// Note:          func(Z,A);
// and            Z=func(A);
// are equivalent.  And fnArgs() is 2 in both cases.
```

1-2

•MCL is a very powerful language. It is very flexible and allows for significant parameterization. This enables very high levels of design reuse.

•The FIR() function above describes a high sample-rate FIR filter that can be re-used for any number of taps and any width/format for the data and coefficients.

- •Input, X, can be any width, signed/unsigned
- •Coefficients can be variable, constant, any width, signed/unsigned
- •Number of taps is determined automatically with the fnArgs() function
- •All of these IO characteristics are determined based on the function ***call*** to FIR()

•MC will automatically synthesize Canonical Signed-Digit multipliers for any constant coefficient multiplier.

- •This does not rely on gate level optimization which would not produce the best results
- •If the constant happens to be a power of 2, there will be only one partial product. This is simply a degenerate case of a CSD multiplier.